APPENDIX A

```
// WinCorr32PktPC.cpp - Win32 PocketPC GUI version
// File Name: WinCorr32PktPC.cpp

#define WIN32_LEAN_AND_MEAN
#include <windows.h>
#include <voicectl.h>
#include "resource.h"
#include "winaudio2.h"

#define MY_REC_TIME 5000  //milisecs

#define NOISE_CORRECTION 3.5  //3.75 //4.10
#define B_CALC_THRES  TRUE
#define USELESS_NOISE_SAMPLE_THRES 0x4000

#define NOISE_THRES        0.18f
#define MAX_FILES_CHOP     3 //10
#define MIN_SAMPLES_CHOP   50
#define MAX_SAMPLES_CHOP   130
#define MIN_SAMPLES_REJECT_CHOP    800 //400
#define SAMPLES_PER_SEC    44100
#define BITS_PER_SAMPLE    16

#define SCALE_FACTOR  0x7FFF
#define NORM( s) ( (real_t)s / SCALE_FACTOR)

#define B_CHOP_CHK  TRUE
#define CORR_THRES  0.6

typedef double real_t;

#if BITS_PER_SAMPLE == 8
      typedef BYTE sample_t;        //8-bit PCM encoded samples
#elif BITS_PER_SAMPLE == 16
      typedef short sample_t;        //16-bit (signed) PCM encoded
samples
#endif

TCHAR szApp[] = TEXT("Termite Detector");
TCHAR szLibFolder[] = TEXT("termite_lib\\");
TCHAR szRecFile[] = TEXT("tdtor_snd.wav");
TCHAR szReportFile[] = TEXT("tdtor_rep.txt");
char szConfigFile[] = "tdtor_cfg.txt";
char szCorrThresKey[] = "corr_thres";

HWND hwndMain, hwndVoice;
FILE *fReport;
real_t corr_thres;

TCHAR ascii_cnt[ 8], chunkfile[ 256], szDBaseFile[ 256];
WIN32_FIND_DATA fdata;

//
//Function declarations
//

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPTSTR lpCmdLine, int nCmdShow);
BOOL CALLBACK WinCorrDlgProc(HWND hWndDlg, UINT msg,
                             WPARAM wParam, LPARAM lParam);
void Handle_WM_NOTIFY( WPARAM wParam, LPARAM lParam);
int MyGetProfileString( LPCSTR sFilename, LPCSTR sKey, LPSTR sValue);
void RecordAudio();
void ProcessFile();
```

```
void Chop();
void Cross();

//
//Function definitions
//

#include "winaudio2.c"

//creates main popup dialog box
//
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPTSTR lpCmdLine, int nCmdShow)

{
  return DialogBox( hInstance, MAKEINTRESOURCE( WINCORR32DLG),
                    0, (DLGPROC)WinCorrDlgProc);
}

//main dialog procedure dispatcher..
//
BOOL CALLBACK WinCorrDlgProc(HWND hWndDlg, UINT msg,
                             WPARAM wParam, LPARAM lParam)

{
  switch ( msg)
  {
    case WM_INITDIALOG:
#if 0
      MessageBox( hwndMain, TEXT("WM_INITDIALOG"), szApp, MB_OK);
#endif
      hwndMain = hWndDlg;
      {
        char sCorrThres[ 10];
        switch (MyGetProfileString( szConfigFile, szCorrThresKey,
sCorrThres) )
        {
        case 0:
        case -1:
          MessageBox( hwndMain, TEXT("Config file or 'corr_thres' key
NOT found, defaulting to 0.6"), szApp, MB_OK);
          corr_thres = CORR_THRES;
          {
            FILE *f = fopen( szConfigFile, "a");
            if (!f) break;
            fprintf( f, "\n%s = %s\n", szCorrThresKey, "0.6");
            fclose( f);
          }
          break;
        case 1:
          corr_thres = atof( sCorrThres);
        }//switch
      }
      return TRUE;

    case WM_COMMAND:
#if 0
      MessageBox( hwndMain, TEXT("WM_COMMAND"), szApp, MB_OK);
#endif
      if ( LOWORD( wParam) == IDC_BTN_DETECT)  //user clicked "detect"
        RecordAudio();
      return TRUE;

    case WM_NOTIFY:  //VoiceRecorder sent msg to owner wnd (main
dialog)
#if 0
```

```
        MessageBox( hwndMain, TEXT("WM_NOTIFY"), szApp, MB_OK);
#endif
        Handle_WM_NOTIFY( wParam, lParam);
        return TRUE;

    case WM_TIMER:  //timer expired
        KillTimer( hwndMain, 1);
#if 1
        MessageBox( hwndMain, TEXT("WM_TIMER"), szApp, MB_OK);
#endif
        SendMessage( hwndMain, VRM_OK, 0, 0);  //finish rec and save file
        return TRUE;

    case WM_CLOSE:
#if 0
        MessageBox( hwndMain, TEXT("WM_CLOSE"), szApp, MB_OK);
#endif
        EndDialog( hWndDlg, 0);
    }
    return FALSE;
}

//handles messages from VoiceRecorder..
//
void Handle_WM_NOTIFY( WPARAM wParam, LPARAM lParam)
{
    NMHDR *pnmh = &((NM_VOICE_RECORDER *)lParam)->hdr;
    switch ( pnmh->code)
    {
    case VRN_RECORD_START:
//      MessageBox( hwndMain, TEXT("VRN_RECORD_START"), szApp, MB_OK);
        break;
    case VRN_RECORD_STOP:
//      MessageBox( hwndMain, TEXT("VRN_RECORD_STOP"), szApp, MB_OK);
//      SendMessage( hwndVoice, VRM_OK, 0, 0);  //finish recording and
save file..
        break;
    case VRN_OK:
//      MessageBox( hwndMain, TEXT("VRN_OK"), szApp, MB_OK);
        ProcessFile();  //recording completed, now process file..
        break;
    }
    return;
}

int MyGetProfileString( LPCSTR sFilename, LPCSTR sKey, LPSTR sValue)
{
    FILE *f;
    int r;
    char sReadKey[ 64];

    f = fopen( sFilename, "r");
    if (f == NULL)
        return 0;  //0: file not found
    for (; ; )
    {
        if (fscanf( f, " %[A-Z_a-z0-9] = %s ", sReadKey, sValue) != 2)
        {
            r = -1;  //-1: key not found
            break;
        }
        if (strcmp( sKey, sReadKey) == 0)
        {
            r = 1;  //1: key found
```

```
        break;
      }
    }
    fclose( f);
    return r;
}

void RecordAudio()
{
  // initialize the VR control struc and create it..
  CM_VOICE_RECORDER cmvr;
  memset( &cmvr, 0, sizeof( CM_VOICE_RECORDER) );
  cmvr.cb = sizeof( CM_VOICE_RECORDER);
//  cmvr.dwStyle = VRS_NO_MOVE; // | VRS_NO_OK;
  cmvr.xPos = 0;  // Use -1 to center the control relative to owner.
  cmvr.yPos = 0;
  cmvr.hwndParent = hwndMain;
  cmvr.lpszRecordFileName = szRecFile;
  hwndVoice = VoiceRecorder_Create( &cmvr);
  if (hwndVoice = NULL)
  {
    MessageBox( hwndMain, TEXT("VoiceRecorder_Create() failed"), NULL,
MB_ICONERROR);
    EndDialog( hwndMain, 0);
  }

  // tell VR to start recording..
//  MessageBox( hwndMain, TEXT("click to send VRM_RECORD"), szApp,
MB_OK);
//  SendMessage( hwndVoice, VRM_RECORD, 0, (LPARAM)szRecFile);
//  MessageBox( hwndMain, TEXT("VRM_RECORD sent"), szApp, MB_OK);

  // set a timer so that recording stops automatically..
/*  if (SetTimer( hwndMain, 1, MY_REC_TIME, NULL) == 0)
  {
    MessageBox( hwndMain, TEXT("SetTimer() failed"), NULL,
MB_ICONERROR);
    EndDialog( hwndMain, 0);
  }*/
}

void ProcessFile()
{
  Chop();
  Cross();

//  Sleep(2000);
//  MessageBox( hwndMain, TEXT("No detection. Try again"), szApp,
MB_OK);
}

class Signal
{
public:

  sample_t *buff;
  UINT n;  //number of sampleS (not bytes!)
  real_t avg;
  real_t sum2;
  sample_t min;
  sample_t max;

  Signal( void *_buff, UINT _n)
  {
```

```
        buff = (sample_t *) _buff;
        n = _n;
        init();
    }

    void init()
    {
        min = 0;
        max = 0;
        avg = 0;
        sum2 = 0;
      UINT i;

        for (i = 0; i < n; i++)
        {
       //avg = ((avg2 * i) + N( buff, i)) / (i + 1);
          avg += NORM( buff[ i]);
          {
              real_t prod = NORM( buff[ i]) * NORM( buff[ i]);
              sum2 += prod;
          }
          if (max < buff[ i])
              max = buff[ i];
          if (buff[ i] < min)
              min = buff[ i];
        }
        avg /= n;
    }//init()
};//class Signal

//Chop()
//
void Chop()
{
  DWORD nRecSize;
  BYTE *pRecBuff = NULL;
  UINT file_cnt = 0;
//    FILE *fReport;

  fReport = _tfopen( szReportFile, TEXT("w") );
  if (fReport == NULL)
      {
          MessageBox( hwndMain, TEXT("Can't create report file (share
violation?)"), NULL, MB_ICONSTOP);
          return;
      }

      _ftprintf( fReport, TEXT("\n*** CHOP REPORT ***\n\n") );

  _ftprintf( fReport, TEXT("Reading \"%s\".. "), szRecFile);

  pRecBuff = (BYTE *)ReadWAVE( szRecFile, &nRecSize, hwndMain);
  if (pRecBuff == NULL)
      {
        _ftprintf( fReport, TEXT("failed!") );
        goto finalize;
      }

      nRecSize /= sizeof( sample_t);  //treat as sample unit (16 bits)

      _ftprintf( fReport, TEXT("done!\nSamples read = %d (%.2f
secs)\n"),
              nRecSize, (float)nRecSize / SAMPLES_PER_SEC);
```

```
        {
          real_t noise_avg = 0.;
          UINT i, j, last_max, first_cut, last_cut;
          int max_max = 0;

          Signal scard( pRecBuff, nRecSize);

    #if 1
        _ftprintf( fReport,
                   TEXT("min = %d\n")
                   TEXT("max = %d\n"),
                   scard.min,
                   scard.max
                 );
    #endif
           for (i = 0; i < scard.n, scard.buff[ i] == 0; i++);  //skip
    first blank samples in .wav

                for ( i = (i == 0)? 1 : i, j = 0;
                          i < scard.n - 1;  //, i < MIN_SAMPLES_CHOP;
                          i++)
                {
                  if (abs( scard.buff[ i - 1]) <= abs( scard.buff[ i]) &&
    //detect maximums
                          abs( scard.buff[ i + 1]) <= abs( scard.buff[ i])
    )
          ;
                  else continue;

          if (abs( scard.buff[ i]) > USELESS_NOISE_SAMPLE_THRES)
            continue;

                  if (max_max < abs( scard.buff[ i]) )
                        max_max = abs( scard.buff[ i]);

                  noise_avg = ((noise_avg * j) + fabs( NORM( scard.buff[
    i]))) / (j + 1);
                   ++j;
                 }//for

        real_t raw_noise_avg = noise_avg;

        noise_avg *= NOISE_CORRECTION * (1 + NOISE_THRES);

    //    MessageBox( hwndMain, TEXT("3"), szApp, MB_OK);

        #if 0
                  _ftprintf( fReport,
                     TEXT("Avg. of max sample values = %.4f\n")
                     TEXT("Thres. correction (user spec) = %.2f\n")
                     TEXT("Internal correction constant = %.4f\n")
                     TEXT("Actual noise level = %.4f\n")
                     TEXT("\n"),
                     raw_noise_avg,
                     NOISE_THRES,
                     NOISE_CORRECTION,
                     noise_avg);
        #else
                  _ftprintf( fReport,
                     TEXT("Avg. of max sample values = %.4f\n")
                     TEXT("Thres. correction (user spec) = %.2f\n")
                     TEXT("Actual noise level = %.4f\n")
                     TEXT("\n"),
```

```
                raw_noise_avg * NOISE_CORRECTION,
                NOISE_THRES,
                noise_avg);
#endif
        for (i = 0; i < scard.n, scard.buff[ i] == 0; i++);  //skip
first blank samples in .wav

get_rising_edge:
    for ( last_max = i, i = (i == 0)? 1 : i;
          i < scard.n - 1;
          i++)  //find first cut
        {
              if (abs( scard.buff[ i - 1]) <= abs( scard.buff[ i]) &&
//detect maximums
                       abs( scard.buff[ i + 1]) <= abs( scard.buff[ i])
);
              else continue;

              first_cut = last_max;
              last_max = i;

              if (fabs( NORM( scard.buff[ i])) < noise_avg)
                continue;

get_falling_edge:
              for (; i < scard.n - 1; i++)  //traverse thru maxs >
noise
              {
                if (abs( scard.buff[ i - 1]) <= abs( scard.buff[ i]) &&
//detect maximums
                         abs( scard.buff[ i + 1]) <= abs( scard.buff[
i]) );
                else continue;

                if (fabs( NORM( scard.buff[ i]) ) < noise_avg
                     && i - first_cut + 1 >= (unsigned)MIN_SAMPLES_CHOP
//add or remove this line
              )
              break;
                }//for

      UINT tmp_cut = i;
                for (;
           i < tmp_cut + 1 + (unsigned)MIN_SAMPLES_REJECT_CHOP && i <
scard.n - 1;
            i++)
                {
                  if (abs( scard.buff[ i - 1]) <= abs( scard.buff[ i]) &&
//detect maximums
                           abs( scard.buff[ i + 1]) <= abs( scard.buff[ i])
);
                  else continue;

                  if (fabs( NORM( scard.buff[ i]) ) > noise_avg)

          {
            if (i - first_cut + 1 < (unsigned)MAX_SAMPLES_CHOP)
              goto get_falling_edge;
            goto get_rising_edge;
          }
        }
```

```
      i = tmp_cut;  //no problem, discard what was done after BEGIN
change

#if 0
              for (; i < scard.n - 1; i++)  //find last cut
              {
                if (abs( scard.buff[ i - 1]) <= abs( scard.buff[ i]) &&
//detect maximums
                    abs( scard.buff[ i + 1]) <= abs( scard.buff[
i]) )
          break;
              }
#endif

      last_cut = last_max = i;

              if (i >= scard.n - 1)
                last_cut = scard.n - 1;

              int chunksize = _tcslen( szRecFile) - sizeof(
TEXT(".wav") ) / sizeof( TCHAR) + 1;
              _tcsncpy( chunkfile, szRecFile, chunksize);
              chunkfile[ chunksize] = '\0';

              _tcscat( chunkfile, TEXT("(") );

              _itot( ++file_cnt, ascii_cnt, 10);
              _tcscat( chunkfile, ascii_cnt);

              _tcscat( chunkfile, TEXT(").wav") );

              if (first_cut >= last_cut)
              {
                _ftprintf( fReport, TEXT("WAV file inconsistency\n") );
                goto finalize;
              }

              DWORD size2write = (last_cut - first_cut + 1) * sizeof(
sample_t);

              _ftprintf( fReport, TEXT("Writing \"%s\" at (begin, end)
= (%d, %d)... "),
                  chunkfile, first_cut, last_cut);

              if (WriteWAVE( chunkfile, size2write, scard.buff +
first_cut, hwndMain) ==  FALSE)
              {
                _ftprintf( fReport, TEXT("failed!\n") );
              }
              else
                _ftprintf( fReport, TEXT("OK\n") );

              first_cut = last_cut + 1;

              if (file_cnt >= MAX_FILES_CHOP)
              {
                _ftprintf( fReport, TEXT("Maximum # of files reached
(%d files).\n"),
                                    MAX_FILES_CHOP);
                break;
              }

        } //outter 'for'
```

```
        if (file_cnt == 0)
          _ftprintf( fReport, TEXT("No chopping done\n") );

        //get rid of chopped files from previous sessions..
        {
                TCHAR chunkfile[ 512], ascii_cnt[ 8];
                int chunksize = _tcslen( szRecFile) - sizeof(
    TEXT(".wav") ) / sizeof( TCHAR) + 1;

          do
          {
                  _tcsncpy( chunkfile, szRecFile, chunksize);
                  chunkfile[ chunksize] = '\0';

                  _tcscat( chunkfile, TEXT("(") );

                  _itot( ++file_cnt, ascii_cnt, 10);
                  _tcscat( chunkfile, ascii_cnt);

                  _tcscat( chunkfile, TEXT(").wav") );

          } while ( DeleteFile( chunkfile) );
        }
    //    ShellExecute( hwndMain, NULL, szReportFile, NULL, NULL, SW_SHOW);
    //open report file
        }

    finalize:

        if (pRecBuff)
          GlobalFree( pRecBuff);

      _ftprintf( fReport, TEXT("\n*** END OF CHOP REPORT ***\n") );

        if (fReport)
             fclose( fReport);

    }//chop()

    //Cross()
    //
    void Cross()
    {
      DWORD nRecSize, nDBaseSize;
      BYTE *pRecBuff = NULL;
      BYTE *pDBaseBuff = NULL;
      UINT file_cnt;
    //  FILE *fReport;

    //  MessageBox( hwndMain, TEXT("Cross() began"), szApp, MB_OK);

      fReport = _tfopen( szReportFile, TEXT("a") );
      if (fReport == NULL)
          {
                MessageBox( hwndMain, TEXT("Can't create report file (share
    violation?)"), NULL, MB_ICONERROR);
            return;
          }

      _ftprintf( fReport, TEXT("\n*** CROSS REPORT ***\n\n") );

          _tcscpy( chunkfile, szRecFile);
```

```
    UINT old_len = _tcslen( chunkfile) - sizeof( TEXT(".wav") ) /
sizeof( TCHAR) + 1;

  for (file_cnt = 1; ; file_cnt++)  //  chopped files loop
  {
    if (B_CHOP_CHK)
    {
      chunkfile[ old_len] = '\0';
            _tcscat( chunkfile, TEXT("(") );
            _itot( file_cnt, ascii_cnt, 10);
            _tcscat( chunkfile, ascii_cnt);
            _tcscat( chunkfile, TEXT(").wav") );
    }

    pRecBuff = (BYTE *)ReadWAVE( chunkfile, &nRecSize, hwndMain);
    if (pRecBuff == NULL)
      break;  //means no more chopped files (or severe error)

    Signal scard( pRecBuff, nRecSize / sizeof( sample_t) );
    int len;

    //get path and append a trailing backslash if needed
    _tcscpy( szDBaseFile, szLibFolder);
    len = _tcslen( szDBaseFile);

    //append wildcards for "find" functions
    _tcscpy( szDBaseFile + len, TEXT("*.wav") );

    HANDLE hfind = FindFirstFile( szDBaseFile, &fdata);  //obtain file
size, attributes, etc.
    if (hfind == INVALID_HANDLE_VALUE)
    {
      MessageBox( hwndMain, TEXT("\"termite_lib\" database folder not
found!"), NULL, MB_OK);
      _ftprintf( fReport, TEXT("FindFirstFile() failed on \"%s\"\n"),
szDBaseFile);
      goto finalize;
    }

    BOOL file_found = FALSE;

    do
    {
      if (fdata.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY)
        continue;

      file_found = TRUE;

      _tcscpy( szDBaseFile + len, fdata.cFileName);

      pDBaseBuff = (BYTE *)ReadWAVE( szDBaseFile, &nDBaseSize,
hwndMain);
      if (pDBaseBuff == NULL)
      {
          _ftprintf( fReport, TEXT("Reading database file \"%s\"
failed!\n"), fdata.cFileName);
        goto finalize;
      }

      Signal dbase( pDBaseBuff, nDBaseSize / sizeof( sample_t) );

          _ftprintf( fReport, TEXT("Cross-Correlating \"%s\" vs.
\"%s\"... "),
                chunkfile, fdata.cFileName);
```

```
        //correlate
        int ns = min( scard.n, dbase.n);
            int nb = max( scard.n, dbase.n);
            sample_t *s = (scard.n < dbase.n) ? scard.buff : dbase.buff;
            sample_t *b = (scard.n >= dbase.n) ? scard.buff : dbase.buff;
            real_t avgsum2 = sqrt( scard.sum2 * dbase.sum2);
        real_t maxcorr;
        real_t corrx;
            int i, k;

        #define Prod( a, b)     ((a) * (b))
        #define avg avgsum2

            maxcorr = 0;
        for (k = 0; k < ns - MIN_SAMPLES_CHOP; k++)
            {
              real_t corr = 0;
          for (i = k; i < ns; i++)
                corr += Prod( NORM( s[ i]), NORM( b[ i - k]) );
              corr /= avg;  //change this as well!!!
          if (maxcorr < corr)
                maxcorr = corr;
            }
        for (k = 1; k < nb - ns + 1; k++)
            {
              real_t corr = 0;
          for (i = 0; i < ns; i++)
                corr += Prod( NORM( s[ i]), NORM( b[ i + k]) );
              corr /= avg;  //change this as well!!!
          if (maxcorr < corr)
                maxcorr = corr;
            }
        for (k = nb - ns + 1; k < nb - MIN_SAMPLES_CHOP; k++)
        {
          real_t corr = 0;
          for (i = k; i < nb; i++)
            corr += Prod( NORM( b[ i]), NORM( s[ i - k]) );
              corr /= avg;  //change this as well!!!
          if (maxcorr < corr)
                maxcorr = corr;
            }
        corrx = maxcorr;

        _ftprintf( fReport, TEXT("%f\n\n"), corrx);

        if (maxcorr > corr_thres)
        {
          MessageBox( hwndMain, TEXT("Insect detected!"), szApp, MB_OK);
          goto finalize;
        }

        if (GlobalFree( pDBaseBuff) == NULL)
          pDBaseBuff = NULL;

//        MessageBox( hwndMain, TEXT("Chopped file crossed"), szApp,
MB_OK);

    } while (FindNextFile( hfind, &fdata) );

    if (GetLastError() != ERROR_NO_MORE_FILES)
    {
      _ftprintf( fReport, TEXT("FindNextFile() failed!\n") );
      goto finalize;
```

```
        }

        if (!file_found)
            _ftprintf( fReport, TEXT("No .wav files found in library!\n")
);

        if (pRecBuff && GlobalFree( pRecBuff) == NULL)
          pRecBuff = NULL;

        if (B_CHOP_CHK == FALSE)
          break;

      }//for (chopped files loop)

      MessageBox( hwndMain, TEXT("No detection. Try again"), szApp, MB_OK);

    //  ShellExecute( m_hWnd, NULL, szReportFile, NULL, NULL, SW_SHOW);
    //open report file

finalize:

      if (pDBaseBuff)
        GlobalFree( pDBaseBuff);

      if (pRecBuff)
        GlobalFree( pRecBuff);

      _ftprintf( fReport, TEXT("*** END OF CROSS REPORT ***\n") );

          if (fReport)
                fclose( fReport);

    }//cross
```

```
//Microsoft Developer Studio generated resource script.
//File Name: WinCorr32PktPC.rc
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
//////////////////////////////////////////////////////////////////////////////
//////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "newres.h"

//////////////////////////////////////////////////////////////////////////////
//////
#undef APSTUDIO_READONLY_SYMBOLS

//////////////////////////////////////////////////////////////////////////////
//////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
//////////////////////////////////////////////////////////////////////////////
//////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""newres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED


//////////////////////////////////////////////////////////////////////////////
//////
//
// Dialog
//

WINCORR32DLG DIALOG DISCARDABLE  0, 0, 142, 63
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "TDtor - Pocket PC ver. 1.0"
FONT 8, "MS Sans Serif"
BEGIN
```

```
    PUSHBUTTON      "DETECT  INSECT",IDC_BTN_DETECT,29,22,84,17
END

/////////////////////////////////////////////////////////////////////////////
//////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    WINCORR32DLG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 135
        TOPMARGIN, 7
        BOTTOMMARGIN, 56
    END
END
#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDI_ICON_MAIN           ICON    DISCARDABLE     "WinCorr32.ico"
#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////
//////


#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//////
//
// Generated from the TEXTINCLUDE 3 resource.
//


/////////////////////////////////////////////////////////////////////////////
//////
#endif    // not APSTUDIO_INVOKED
```

```
//{{NO_DEPENDENCIES}}  File Name :resource.h
// Microsoft Developer Studio generated include file.
// Used by WinCorr32PktPC.rc
//
#define WINCORR32DLG                    101
#define IDI_ICON_MAIN                   102
#define IDC_BTN_DETECT                  1000

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        103
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
//Windows Audio Library v2.0
//File Name: winaudio2.c

#include "winaudio2.h"

void *ReadWAVE( LPCTSTR filename, DWORD *pbSize, HWND hwnd)
{
  HANDLE f = INVALID_HANDLE_VALUE;
  DWORD dwBytesread;
  MY_MMCKINFO *pRiffCkHdr, *pFmtCkHdr, *pDataCkHdr;
  PCMWAVEFORMAT *pWaveHdr;
  void *pSamples;
  BYTE bHdr[ 0x40];

  //open WAVE file
  f = CreateFile( filename, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, 0);
  if (f == INVALID_HANDLE_VALUE)
  {
    if (GetLastError() != ERROR_FILE_NOT_FOUND)
      MessageBox( hwnd, TEXT("Can't open WAVE file"), NULL,
MB_ICONERROR);
    return NULL;
  }

  //read WAVE header
  if (ReadFile( f, bHdr, sizeof( bHdr), &dwBytesread, NULL) == 0
    || dwBytesread != sizeof( bHdr)
    )
  {
    MessageBox( hwnd, TEXT("Can't read WAVE header"), NULL,
MB_ICONERROR);
    CloseHandle( f);
    return NULL;
  }

  pRiffCkHdr = (MY_MMCKINFO *)bHdr;
  pFmtCkHdr = (MY_MMCKINFO *)((DWORD)pRiffCkHdr + 8 + 4);
  pWaveHdr = (PCMWAVEFORMAT *)((DWORD)pFmtCkHdr + 8);
  pDataCkHdr = (MY_MMCKINFO *)((DWORD)pWaveHdr + pFmtCkHdr->cksize);

  //check WAVE headers
  if ( pRiffCkHdr->ckid != mmioFOURCC( 'R','I','F','F')
    || pRiffCkHdr->fccType != mmioFOURCC( 'W','A','V','E')
    || pFmtCkHdr->ckid != mmioFOURCC( 'f','m','t',' ')
    || (DWORD)pDataCkHdr > (DWORD)pRiffCkHdr + sizeof( bHdr) - 8
    || pDataCkHdr->ckid != mmioFOURCC( 'd','a','t','a')
//    || (*(WORD *)&pDataCkHdr->ckid != 'ad' && *((WORD *)&pDataCkHdr-
>ckid + 1) != 'at')
    )
  {
    MessageBox( hwnd, TEXT("Inconsistent or unusual WAVE header"),
NULL, MB_ICONERROR);
    CloseHandle( f);
    return NULL;
  }

  //check WAVE is PCM audio at 44.1K, 16 bits, mono
  if ( pWaveHdr->wf.wFormatTag != WAVE_FORMAT_PCM
    || pWaveHdr->wf.nSamplesPerSec != 44100
    || pWaveHdr->wBitsPerSample != 16
    || pWaveHdr->wf.nChannels != 1
//    || pWaveHdr->wf.nAvgBytesPerSec != 88200
//    || pWaveHdr->wf.nBlockAlign != 2
```

```
    )
  {
    MessageBox( hwnd, TEXT("WAVE file is not PCM at 44.1K, 16 bits,
mono"), NULL, MB_ICONERROR);
    CloseHandle( f);
    return NULL;
  }

  //allocate memory to hold WAVE data
  pSamples = GlobalAlloc( GPTR, pDataCkHdr->cksize);
  if (pSamples == NULL)
  {
    MessageBox( hwnd, TEXT("Can't allocate memory"), NULL,
MB_ICONERROR);
    CloseHandle( f);
    return NULL;
  }

  //read WAVE data
  if ( SetFilePointer( f, (DWORD)pDataCkHdr + 8 - (DWORD)pRiffCkHdr,
NULL, FILE_BEGIN) == -1
    || ReadFile( f, pSamples, pDataCkHdr->cksize, &dwBytesread, NULL)
== 0
//    || dwBytesread != pDataCkHdr->cksize
    )
  {
    MessageBox( hwnd, TEXT("Can't read WAVE data"), NULL,
MB_ICONERROR);
    GlobalFree( pSamples);
    CloseHandle( f);
    return NULL;
  }

  *pbSize = dwBytesread & -2; //pDataCkHdr->cksize;
  CloseHandle( f);
  return pSamples;
}//ReadWAVE()

//WriteWAVE()
//
BOOL WriteWAVE( LPCTSTR filename, DWORD bSize, void *pSamples, HWND
hwnd)
{
  HANDLE f = INVALID_HANDLE_VALUE;
  DWORD dwByteswritten;
  MY_MMCKINFO *pRiffCkHdr, *pFmtCkHdr, *pDataCkHdr;
  PCMWAVEFORMAT *pWaveHdr;
  BYTE bHdr[ 8 + 4 + 8 + sizeof( PCMWAVEFORMAT) + 8];  //0x2C

  //create WAVE file
  f = CreateFile( filename, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, 0);
  if (f == INVALID_HANDLE_VALUE)
  {
    MessageBox( hwnd, TEXT("Can't create WAVE file"), NULL,
MB_ICONERROR);
    return FALSE;
  }
  //create WAVE headers...

  pRiffCkHdr = (MY_MMCKINFO *)bHdr;
  pFmtCkHdr = (MY_MMCKINFO *)((DWORD)pRiffCkHdr + 8 + 4);
  pWaveHdr = (PCMWAVEFORMAT *)((DWORD)pFmtCkHdr + 8);
```

```
    pDataCkHdr = (MY_MMCKINFO *)((DWORD)pWaveHdr + sizeof( PCMWAVEFORMAT)
);

    pRiffCkHdr->ckid = mmioFOURCC( 'R','I','F','F');
    pRiffCkHdr->cksize = (8 - 8) + 4 + 8 + sizeof( PCMWAVEFORMAT) + 8 +
bSize;
    pRiffCkHdr->fccType = mmioFOURCC( 'W','A','V','E');

    pFmtCkHdr->ckid = mmioFOURCC( 'f','m','t',' ');
    pFmtCkHdr->cksize = sizeof( PCMWAVEFORMAT);

    pWaveHdr->wf.wFormatTag = WAVE_FORMAT_PCM;
    pWaveHdr->wf.nSamplesPerSec = 44100;
    pWaveHdr->wBitsPerSample = 16;
    pWaveHdr->wf.nChannels = 1;
    pWaveHdr->wf.nAvgBytesPerSec = 88200;
    pWaveHdr->wf.nBlockAlign = 2;

    pDataCkHdr->ckid = mmioFOURCC( 'd','a','t','a');
    pDataCkHdr->cksize = bSize;

    //Write WAVE header and data
    if ( WriteFile( f, bHdr, sizeof( bHdr), &dwByteswritten, NULL) == 0
      || dwByteswritten != sizeof( bHdr)
      || WriteFile( f, pSamples, bSize, &dwByteswritten, NULL) == 0
      || dwByteswritten != bSize
      )
    {
      MessageBox( hwnd, TEXT("Can't write WAVE file"), NULL,
MB_ICONERROR);
      CloseHandle( f);
      return FALSE;
    }
    CloseHandle( f);
    return TRUE;
}//WriteWAVE()
```

```
//Windows Audio Library v2.0
//File Name : winaudio2.h

#ifndef _WINAUDIO2_H
#   define _WINAUDIO2_H

#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <mmsystem.h>

typedef     DWORD MY_FOURCC;

// RIFF MultiMedia Chunk information data structure
// (Note: not present in Windows CE from Pocket PC)
typedef struct
{
      MY_FOURCC       ckid;           /* chunk ID */
      DWORD           cksize;         /* chunk size */
      MY_FOURCC       fccType;        /* form type or list type */
      DWORD           dwDataOffset;   /* offset of data portion of
chunk */
      DWORD           dwFlags;        /* flags used by MMIO functions
*/
} MY_MMCKINFO;

void *ReadWAVE( LPCTSTR filename, DWORD *pbSize, HWND hwnd);
BOOL WriteWAVE( LPCTSTR filename, DWORD bSize, void *pSamples, HWND
hwnd);

#endif //_WINAUDIO2_H
```

```
// File Name: newres.h

#ifndef __NEWRES_H__
#define __NEWRES_H__

#if !defined(UNDER_CE)
#define UNDER_CE _WIN32_WCE
#endif

#if defined(_WIN32_WCE)
      #if !defined(WCEOLE_ENABLE_DIALOGEX)
            #define DIALOGEX DIALOG DISCARDABLE
      #endif
      #include <commctrl.h>
      #define  SHMENUBAR RCDATA
      #if defined(WIN32_PLATFORM_PSPC) && (_WIN32_WCE >= 300)
            #include <aygshell.h>
            #define AFXCE_IDR_SCRATCH_SHMENU  28700
      #else
            #define I_IMAGENONE           (-2)
            #define NOMENU                0xFFFF
            #define IDS_SHNEW       1

            #define IDM_SHAREDNEW         10
            #define IDM_SHAREDNEWDEFAULT 11
      #endif // _WIN32_WCE_PSPC
      #define AFXCE_IDD_SAVEMODIFIEDDLG 28701
#endif // _WIN32_WCE

#ifdef RC_INVOKED
#ifndef _INC_WINDOWS
#define _INC_WINDOWS
      #include "winuser.h"            // extract from windows header
      #include "winver.h"
#endif
#endif

#ifdef IDC_STATIC
#undef IDC_STATIC
#endif
#define IDC_STATIC      (-1)

#endif //__NEWRES_H__
```